МІНІСТЕРСТВО  ОСВІТИ  І  НАУКИ  УКРАЇНИ

НАЦІОНАЛЬНИЙ   ТЕХНІЧНИЙ   УНІВЕРСИТЕТ   УКРАЇНИ

“КИЇВСЬКИЙ  ПОЛІТЕХНІЧНИЙ  ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 5

з дисципліни “Програмування”

тема “Поведінкові шаблони”

|  |  |  |
| --- | --- | --- |
| Виконав(ла)  студент(ка) II курсу  групи КП-01  Грабовська Анастасія Дмитрівна  (*прізвище, ім’я, по батькові*)  Варіант - 5 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Заболотня Тетяна Миколаївна  (*прізвище, ім’я, по батькові*) |

Київ 2021

**Мета роботи**

Ознайомитися з поведінковими шаблонами у C# та реалізувати дві задачі за допомогою цих паттернів.

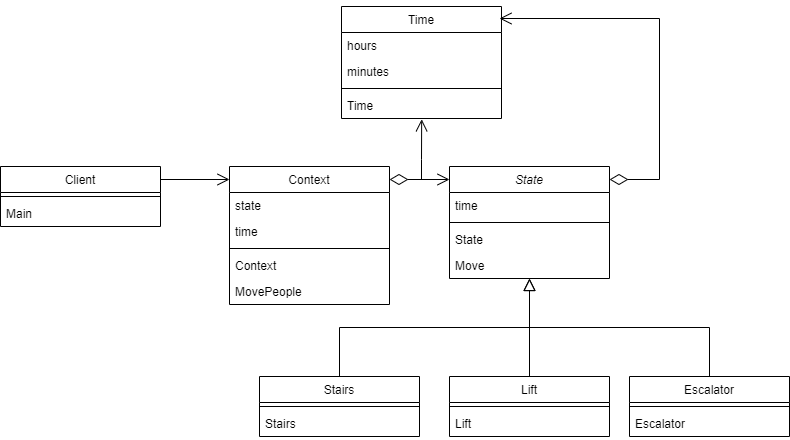
**Постановка задачі**

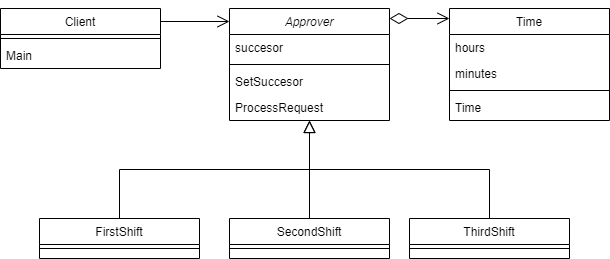
1. За допомогою шаблону проєктування реалізувати декілька способів переміщення нагору та вниз у торговому центрі. Наприклад, можна ходити сходами, їздити на ескалаторі та їздити на ліфті. В залежності від стану устаткування або від часу доби активувати той чи інший спосіб пересування відвідувачів. Наприклад, з 22.00 до 10.00 кожного дня ескалатори та ліфти перестають працювати. А під час профілактичних робіт з ескалатором має працювати ліфт та навпаки.
2. На певному заводі цілодобово працюють 3 зміни. І зміна – з 6.00 до 15.00, ІІ зміна – з 14.00 до 23.00 та ІІІ зміна – з 22.00 до 7.00. Якщо завдання робітникам певної зміни надходить менш, ніж за годину до закінчення зміни, вони передають його на виконання наступній зміні. За допомогою шаблону проєктування реалізувати механізм обробки завдань робітниками описаного заводу.

**Обгрунтування вибору шаблону**

1. Очевидно, що для цієї структури класів потрібно реалізувати шаблон **стан**, оскільки потрібно динамічно міняти поведінку об’єкта в залежності від контексту. Поведінки, що залежать від стану, переїжджають в окремі класи. Початковий клас зберігає посилання на одного з таких об’єктів-станів і делегує йому роботу.
2. Очевидно, що для цієї структури класів потрібно реалізувати шаблон **ланцюжок обов’язків (Chain Of Responsibilities),** оскільки ми маємо кілька груп виконавців, до яких надходять завдання, і, в залежності від умов, потрібно вирішити хто буде виконавцем. Всі можливі обробники запиту утворюють ланцюжок, а сам запит переміщається цим ланцюжком. Кожен об'єкт у цьому ланцюжку при отриманні запиту вибирає або закінчити обробку запиту або передати запит на обробку наступному по ланцюжку об'єкту.

**UML діаграма класів**





**Код програми:**

5.

|  |
| --- |
| abstract class State  {      protected Time time;      public State(Time time)      {          this.time = time;      }      public abstract void Move();  }  class Stairs : State  {      public Stairs(Time time) : base(time)      {}      public override void Move()      {          Console.WriteLine($"Move by stairs. Time is {time.Hours}:{time.Minutes}, lift and escalator is closed.");      }  }  class Lift : State  {      public Lift(Time time) : base(time)      {}      public override void Move()      {          Console.WriteLine($"Move by lift. Time is {time.Hours}:{time.Minutes}, escalator is closed.");      }  }  class Escalator : State  {public Escalator(Time time) : base(time)      {}      public override void Move()      {          Console.WriteLine($"Move by escalator. Time is {time.Hours}:{time.Minutes}, lift is closed.");      }  }  class Context  {      State state;      Time time;      public Context(Time time)      {          this.time = time;          if(time.Hours <= 10 || time.Hours >= 22) state = new Stairs(time);          if(time.Hours > 10 && time.Hours < 14) state = new Lift(time);          if(time.Hours >= 14 && time.Hours < 22) state = new Escalator(time);      }      public void MovePeople()      {          this.state.Move();      }  }  class Time  {      public int Hours {get; private set;}      public int Minutes {get; private set;}      public Time(int hours, int minutes)      {          Hours = hours;          Minutes = minutes;      }  } |
| Time time1 = new Time(4, 20);  Time time2 = new Time(11, 54);  Time time3 = new Time(19, 12);  Context context1 = new Context(time1);  Context context2 = new Context(time2);  Context context3 = new Context(time3);  context1.MovePeople();  context2.MovePeople();  context3.MovePeople(); |

6.

|  |
| --- |
| abstract class Approver  {      protected Approver successor;      public void SetSuccessor(Approver successor)      {          this.successor = successor;      }      public abstract void ProcessRequest(Time time);  }  class FirstShift : Approver  {      public override void ProcessRequest(Time time)      {          if (time.Hours >= 6 && time.Hours < 14)          {              Console.WriteLine($"{this.GetType().Name} approved request. This request came at {time.Hours}:{time.Minutes}.");          }          else if (successor != null)          {              successor.ProcessRequest(time);          }      }  }  class SecondShift : Approver  {      public override void ProcessRequest(Time time)      {          if (time.Hours >= 14 && time.Hours < 22)          {              Console.WriteLine($"{this.GetType().Name} approved request. This request came at {time.Hours}:{time.Minutes}.");          }          else if (successor != null)          {              successor.ProcessRequest(time);          }      }  }  class ThirdShift : Approver  {      public override void ProcessRequest(Time time)      {          if (time.Hours >= 22 || time.Hours < 6)          {              Console.WriteLine($"{this.GetType().Name} approved request. This request came at {time.Hours}:{time.Minutes}.");          }          else if (successor != null)          {              successor.ProcessRequest(time);          }      }  }  class Time  {      public int Hours {get; private set;}      public int Minutes {get; private set;}      public Time(int hours, int minutes)      {          Hours = hours;          Minutes = minutes;      }  } |
| Time time1 = new Time(4, 20);  Time time3 = new Time(19, 12);  Time time4 = new Time(16, 55);  Approver firstShift = new FirstShift();  Approver secondShift = new SecondShift();  Approver thirdShift = new ThirdShift();  firstShift.SetSuccessor(secondShift);  secondShift.SetSuccessor(thirdShift);  thirdShift.SetSuccessor(firstShift);  firstShift.ProcessRequest(time1);  secondShift.ProcessRequest(time3);  thirdShift.ProcessRequest(time4); |

**Приклади результатів:**

5. Console:



6. Console:



**Висновки:**

Під час виконання лабораторної роботи я ознайомилася з поведінковими шаблонами у C# та реалізувала дві задачі за допомогою цих паттернів.